OFL : How to capture and customise the operational semantics of object languages based on classes
Team OCL of I3S laboratory

Philippe LAHIRE

Workshop 02.13.02 – Cambridge – Microsoft research
Summary

• About the team and research topics
• Motivation
  – The problem to solve
  – Intuitive approach
• Overview of OFL model
  – Definition
  – Possible use
  – Results
• Possible cooperation
Team OCL of I3S Laboratory

• I3S: computer science laboratory of the university of Nice – Sophia Antipolis

• OCL Team:
  – **Background**: Software engineering, Object-oriented languages and Database management
  – **Assertion and programming by contract**
  – **Meta-modelling of object language semantics and interoperability**

About the team : our research interests
Context

• Object languages based on classes
  – C#
  – Java
  – Eiffel
  – C++

• Software engineering
  – Focus on controls
  – Pragmatic approach
The road to meta modelling

- To include views in object languages
- Views are only a specific use of inheritance
- To distinguish the use of views from the other usages of inheritance mechanism
- To create a specific relationship for views
- To define an object meta model which allows to create relationships for views but also for other uses ...
Objectives

• Improve source code and language interoperability thru a more accurate specification of relationship
  – More readability and controls
  – More accurate documentation
  – A support to include application additional concerns
• To provide this better use specification but with no obligation to use it
• To be able to adapt a language in the framework defined above
Our approach: meta programming

- To create new types of relationships
- To create new types of classes
- To integrate them within the scope of a language
  - In order to create a new language or,
  - Much more probably, to adapt an existing one.
Meta programming : OFL philosophy

→ To reify relationships as well as classes
→ To make them embedded language elements
→ To provide a simple way to create those components and to limit complex source code writing, needed for its implementation.
→ When the access to source code is mandatory then to locate user intervention
OFL: *Open Flexible Languages*

- Three essential elements
  - Component-description
    - Customisable meta classes
  - Component-relationship
    - Customisable meta relationship
  - Component-language
    - Meta language, composition
Semantics customisation : parameters

- Each component is mainly described with a set of parameters.
- Parameter ≈ part of semantics
- Creation of a component by specialisation of a generic component
- Under meta programmer responsibility
- Control of components with assertion mechanism
Semantics customisation: entry points

- It is necessary to take the definition of a component with the valuation of its parameters in order to describe the system operational semantics.
- OFL defines a system of 53 actions (entry points).
- Action ≈ algorithm which defines a part of application execution
- Each action takes into account the value of the parameters which influences its execution
- Examples of actions:
  - Message sending
  - Feature lookup (dynamic binding)
  - Assignment of an object instance to an attribute
Architecture d’OFL

Generic concepts

Language components

An application

The OFL model: Overview
Hierarchy of concepts

The OFL model: Overview
Part of the hierarchy of elementary components

The OFL model: overview
Some of the 13 parameters of a concept-description

- Name ("Class", "Interface" or ...)
- Generator (true or false)
- Sharing_control (description, instance or unique_instance)
- Attribute (allowed or forbidden)
- Method (allowed or forbidden)
- Overloading (allowed or forbidden)
  - attributes,
  - results of function,
  - Number of method parameters and
  - Type of method parameters
Some of the 28 parameters of a concept-relationship

- **Name** ("Inheritance", "Code_Reuse", "View", ...)
- **Cardinality** (1-1, 1-∞, or ...)
- **Repetition** (allowed or forbidden for sources & targets)
- **Circularity** (allowed or forbidden)
- **Opposite** ("Generalisation", "View", ...)
- **Polymorphism_direction** (none, up, down or both)
- **Polymorphism** (hiding or overriding for attributes & methods)
- **Feature_variance** (covariant, contravariant, nonvariant or non_applicable for method parameters, function results and attributes)
- **Renaming** (mandatory, allowed or forbidden)
One action among 53

• Feature lookup(M : Message)
  • Some parameters of concept-description which are handled:
    ➢ Generator
    ➢ Overloading
  • Some parameters of concept-relation which are handled:
    ➢ Cardinality
    ➢ Circularity
    ➢ Opposite
    ➢ Polymorphism_direction
    ➢ Polymorphism
    ➢ Feature_variance
    ➢ Renaming
Methodology for building the OFL semantics of a language

- **To create components-descriptions**
  - To instanciate a concept-description & to specialise a component-description
  - To provide a value for each parameter in order to define the semantics

- **To create components-relationships**
  - To instanciate a concept-relationship & to specialise a component-relationship
  - To provide a value for each parameter in order to define the semantics

- **To create one component-language**
  - To instanciate a concept-language & to specialise a component-language
  - To integrate in it components-descriptions & components-relationships
Why Generalisation and Specialisation relationships?

↑ specialisation  ↓ generalisation

The OFL model: Overview
To reuse code

class Person {

    int yearOfBirth;

    int age() {
        return Date.currentYear() - yearOfBirth;
    }
}

class Car code_reuses Person {

    int yearOfFirstUse;

    int age() from Person where yearOfBirth is
    yearOfFirstUse;
}
OFL applied to some other concerns

A better modelling of application but also some other functionalities may be handled:

- Controls and conversion operations between operations carried out by remote objects
- Storage, retrieval and integrity checking of persistent data
- Metrics analysis
- Dynamic quality service / critical situation
- Code generation
- ...

Code of action is adapted to the use of one/several concerns
OFL : Key issues

- Customisation of language
- System of actions that may be redefined
- Each action is a set of several concerns (control, code generation, etc.)
- Control of model consistency and of its uses thru the use of assertions
- To separate tasks: meta-programming & programming
- Model is independent of languages
- A language is a set of components tat may be integrated
- Aim to improve language inter-operability
OFL environement

The OFL model: Overview
Possible common interests?

- Context of XLANG / BizTalk:
  - Customisation of the handling of message exchange or exception handling according to the context (application, language, …)
  - Heterogeneous schema of classes
  - …

- Other context: Toward a better specification of operational semantics dedicated to different purposes.

Perspectives of cooperation
A Major issue: interoperability

Operational semantics
C#

Module of Application For accounting
http://siteNumberOne

Request
Controls, Conversion, ...
Response

Piece of semantics

Piece of semantics

Operational semantics
Java

Module of Application For E-commerce
http://siteNumberTwo

Perspectives of cooperation
OFL : our response

- To better specify inter-description relationships
- To provide but not impose new functionalities
- To encapsulate language behaviour into the application

⇒ To favour language interoperability
⇒ To favour language distribution
⇒ To encapsulate (additional) concerns within languages (persistence, distribution, conversion, etc.)